Advance in Electronic and Electric Engineering.
ISSN 2231-1297, Volume 3, Number 7 (2013), pp. 817-826

http://www.ripublication.com/aeee.htm

Fault Models and Test Generation for Covalidation Techniques in Hardware & Software

Deepti Rajput

Teerthanker Mahaveer University, Moradabad.

Abstract

This paper describes the different types of test generation methods, different type of fault models. Basically it describes the fault models and their hardware software implementation. This paper focuses on the test generation process for hardware software systems as well as the fault models and fault coverage analysis techniques which support test generations. Automatic test generation techniques have been presented which are applicable to large scale designs, but until the underlying fault models are accepted, the techniques will not be applied in practice. A useful techniques Obstacle the widespread acceptance of available techniques is the lack in correlation between covalidation fault models and real design errors. Fault models must be evaluated by identifying a correlation between fault coverage and detection of real design errors. There is mandatory to evaluate the compilation of design errors produced by real designers. Once covalidation fault models are empirically evaluated we can expect to see large increases in covalidation productivity through the automation of test generation.

Keywords: Fault Models, Covalidation Techniques, Design Errors.

1. Introduction

Hardware-software systems are pervasive in the electronics systems industry. The widespread use of these systems in cost-critical and life-critical applications motivates the need for a systematic approach to verify functionality. Several obstacles to the verification of hardware-software systems make this a challenging problem. To manage the complexity of the problem, covalidation techniques in which functionality is verified by simulating (or emulating) a system description with a given test input

sequence are being considered. Hardware-software systems are built from separate components which are not globally synchronized. As a result, hardware-software systems are vulnerable to inter-process synchronization problems resulting from timing problems between processes. In previous work we have developed a fault model to describe these timing-induced errors [1] and we have presented a test generation approach for the fault model [2]. Requirements upon VLSI designs are continuously increasing towards faster and larger circuits, leading to area and timing optimized designs, and raising demands for testing. Testing should be thorough, to confirm high specifications, but should not require area expensive Design for Testability circuitry. This calls for test pattern generation for more realistic fault models as the widely used stuck at fault model, like the delay fault model [3].Hardware verification complexity alone has increased to the cost of design. So to manage the complexity of the problem, many researchers are investigating *covalidation* techniques, in which functionality is verified by simulating (or emulating) a system description with a given test input sequence. In contrast, formal verification techniques have been explored which verify functionality by using formal techniques (i.e. model checking, equivalence checking, automatic theorem proving) to precisely evaluate properties of the design. The tractability of covalidation makes it the only practical solution for many real designs.

2. Fault Models and Coverage Evaluation

The accurateness of testing depends upon an accurate behavioral description of the circuits on chips containing physical failure. A great body of research exists concerning fault models and their applications. The result of early studies the abstraction logic level with the stuck at fault model provides the basis for fault simulation, test generation and other testing analysis application. So its popularity, the stuck at fault model does not describes the behaviour of all physical Failures. Eldred suggested an efficient test generation approach that targets hardware faults rather than the function. This is done by creating test patterns for specific faults. Commonly occurring physical faults are represented by logical fault models. Logical faults represent the effect of physical faults on the behaviour of the digital circuit. Then, input stimuli are created to distinguish between the fault-free and faulty circuits. Test pattern generation based on the logical fault model assumes the presence of a single fault in the circuit at a given time. Test patterns derived under the single-fault assumption are generally considered useful for detecting multiple faults because a test derived for an individual single fault can detect a multiple fault containing that single fault as a component. There are, however, specific multiple faults where the components can mask each other and detection by a single fault test is not guaranteed .The evaluation of covalidation fault models depends on following two individuals 1.accurateness in terms of design defects 2. Efficiency in terms of the number of faults in a design. Hardware software covalidation designs are based on a top down design methodology which begins with a behavioural system description. So the covalidation fault models are behavioural level fault models. Existing covalidation fault models can

be classified by the style of behavioral description upon which the models are based. Textual languages, such as VHDL and ESTEREL are specified the system behaviours and converted into an internal behavioural format for use in co design and cosimulation. Many different internal behavioral formats are possible (1). Hardware software designs have the origins in either the hardware (2) or the software (3) domains and it is currently applied for the covalidation fault models. To explain the covalidation fault models we are using here an example given in figure 1. In this figure there are two parts figure 1a and figure 1b. In figure 1a the behaviour of the system has been shown while in figure 1b the corresponding control-data flow charts (CDFG) has been represented. Figure 1 is composed of only a single process and it there is no signals which are used to model real time in most hardware description languages. There are so many limitations until the example is adequate to describe the relevant characteristics of many covalidation fault model.[4]

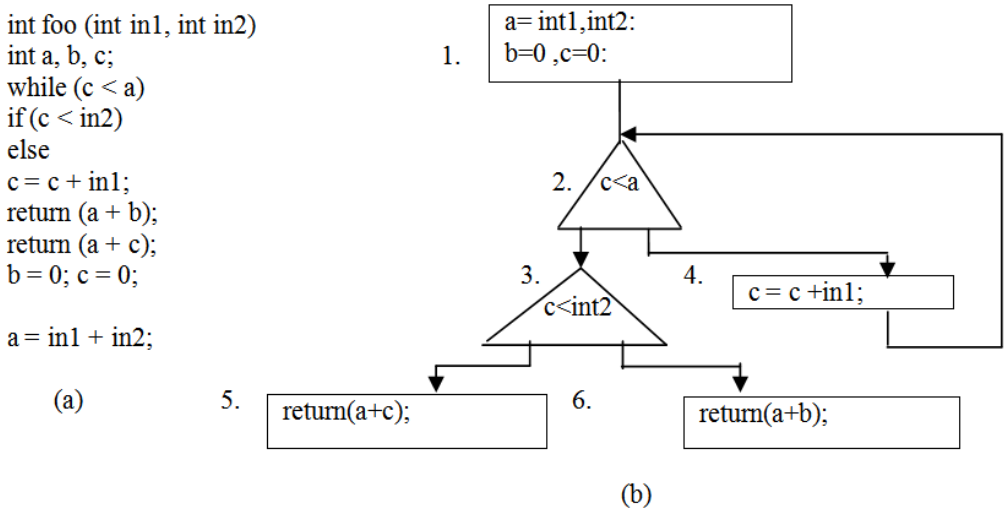

Figure 1: Behavioral Descriptions, (a) Textual Description,
(b) Control-Dataflow Graph (CDFG)

2.1 Textual Fault Modelling

The fault models which are directly applied to original textual behaviour description are known as textual fault model. The simplest textual fault model is the statement coverage metric introduced in software testing [5] which associates a potential fault with each line of code, and requires that each statement in the description be executed during testing. The result of this model is excellent because in this model the number of potential fault is equal to the number of lines of codes. In branch coverage metric the efficiency can be completed by analyzing a single co simulation output trace due to which its efficiency is high. The researchers has been used branch coverage metric for behavioral validation for coverage evaluation and test generation [6,7,8] and some

other researchers have been also studied the accuracy of branch coverage to find its ability to cover design defects.

2.2 Control Data Flow Fault Models

Any automated procedure requires that input data being provided is in some predefined format. Also, the models used to represent the inputs and transformations (changes of the input) should be efficient for execution of the procedure. For example, in case of HLS the input specifications are generally in some Hardware Definition Language (HDSs) like Verilog, VHDL, and System C etc. here we are use the VHDL language. The HDL specifications are represented using several modeling paradigms like Control and Data Flow Diagram (CDFG) , DeJong's hybrid flow graph, SSIM flow graph, Finite state machine with data etc., which are suitable for scheduling, allocation and binding procedures. Sometimes timing constrains (on execution of steps) are also given in the specifications, which are modelled by the above paradigms, however, with timing parameter included e.g., CDFG with timing, DF with timing and CF with timing. CDFG is one of the most widely used modelling paradigm and the others mentioned above are not much different. In general, the nodes in a CDFG can be classified into one of the following types:

- *Operational nodes:* These are responsible for arithmetic, logical or relational operations (or computations); e.g., addition, equality checking etc.
- *Control nodes:* These nodes are responsible for control operations like conditions, loop constructs etc.; e.g., case statements, while loop etc.
- *Storage nodes:* These nodes represent assignment operations associated with variables and signals; e.g., reading an input value to register etc.

The edges in a CDFG represent Transfer of values (in variables that are changed due to processing in Operational and storage nodes). A node needs data generated by its predecessor nodes and generates new data needed by its successors. Nodes operate on the data of the incoming edges. The resulting data is put on the outgoing edges.

- *Control flow* from one node to another: An edge can also represent a condition, e.g., while implementing loop constructs, if/case statements etc. In dataflow testing, each variable occurrence is classified as either a definition occurrence or a use occurrence. Some variables are connected to select Paths. For example node1 in Figure 1b, define the signal a and nodes 2, 5, and 6 contain uses of signal a. And the paths 1, 2, 4, 5 and 1, 2, 4,6 must be executed in order to cover both of these definition-use pairs. The dataflow testing criteria have also been applied to behavioral hardware descriptions [9].

2.3 Gate-Level Fault Models

For decades, traditional IC test generation has been at the gate level based on the gate-level netlist. The stuck-at fault model can easily be applied for which many ATPG and fault simulation tools are commercially available. Very often the stuck at fault model is also employed to evaluate the effectiveness of the input stimuli used for simulation-based design verification. As a result, the design verification stimuli are often also used

for fault detection during manufacturing testing. In addition to the stuck-at fault model, delay fault models and delay testing have been traditionally based on the gate-level description. While bridging faults can be modelled at the gate level, practical selection of potential bridging fault sites requires physical design information. The gate-level description has advantages of functionality and tractability because it lies between the RTL and physical levels; however, it is now widely believed that test development at the gate level is not sufficient for deep submicron designs.

2.4 Stuck At Fault Models

A stuck-at fault is a particular fault model used by fault simulators and automatic test pattern generation (ATPG) tools to mimic a manufacturing defect within an integrated circuit. Individual signals and pins are assumed to be stuck at Logical '1', '0' and 'X'. For example, an output is tied to a logical 1 state during test generation to assure that a manufacturing defect with that type of behavior can be found with a specific test pattern. Likewise the output could be tied to a logical 0 to model the behavior of a defective circuit that canno switch its output pin. Not all faults can be analyzed using the stuck-at fault model. Compensation for static hazards, namely branching signals, can render a circuit untestable using this model. Also, redundant circuits cannot be tested using this model, since by design there is no change in any output as a result of a single fault. Stuck at fault model is as convenient to show the complexity of analyzing multiple fault model. This result in 3^n -1 possible faulty circuit to consider. So the stuck at fault models is time and input variant

2.5 Bridging Faults

Different types of models have been proposed to describe the unintentional connection between two nodes. So the bridging faults stems both the insufficiency occurrence of interconnect shorts. So the bridging fault is dependent upon the technology, the failure mechanism and the target application or simulator. Bridging fault may be modelled as a logical fault, which change the logic value on a node or as an electrical fault so the voltage and current change within the circuit.

2.6 Design Faults

There are three major types of design faults in a system those “inherited” the system, made by human designers, and the other made by the computers that aid in the design process [10]. Inherited faults are existing before starting the design process. For example, conflicting specifications are inherited faults considered. These faults cannot be completely eliminated because no system is completely new. Human design faults two types: data preparation faults and transcription faults. Data preparation faults usually result from making wrong decisions, miscalculations, etc. Transcription faults are transferring data from one medium to another without changing its content. Faults due to mistakes in keying design data into a computer are considered transcription faults. Human design faults must be detected as early as possible because it costs a lot

to detect and correct them later. They can happen at any stage of the design process and can remain undiscovered throughout the lifetime of the system.

2.7 Fabrication Faults

These are not directly involved to human error; instead they are from an imperfect manufacturing process. For example, shorts and opens are common defects in the manufacture of very large-scale integrated (VLSI) circuits using CMOS technology, the industry standard. These defects can have a severe effect on the behavior of an IC. CMOS fabrication defects include incorrect transistor threshold voltage, improper doping profiles, mask alignment errors, and poor encapsulation. Accurate identification of fabrication defects is important in improving the manufacturing yield. [11]

2.8 Operational Faults

Most of the operational faults are caused by external disturbance during the normal operation of the digital system. There are some Common sources of operational faults are electromagnetic interference, operator mistakes, environmental extremes, and wear out. For example, if a digital system is subjected to extreme temperature variations, the system can give us incorrect results. Moreover, excessive temperature and humidity accelerate the aging of components. Some operational faults arise due to the movement of the system, especially in mobile applications. Also, some IC faults are due to electron migration, where metal connectors inside an IC package thin out with time and break. Operator mistakes are considered in this class because an operator may provide incorrect commands which lead to system failure. Operational faults are classified according to their duration:

- *Permanent faults* remain in existence indefinitely if no corrective action is taken. Many of these are residual design or manufacturing faults. Those that are not most frequently occur during changes in system operation, for instance, after system start-up or shutdown, or as a result of a catastrophic environmental disturbance such as a collision.
- *Intermittent faults* appear, disappear, and reappear repeatedly. They are difficult to predict, but their effects are highly correlated. Most intermittent faults are due to marginal design or manufacturing. The system works well most of the time, but fails under atypical environmental conditions.
- *Transient faults* appear and disappear quickly, and are not correlated with each

3. Automatic Test Pattern Generation

Due to the imperfect manufacturing process, defects may be introduced during fabrication, resulting in chips that could potentially malfunction. The objective of test generation is the task of producing a set of test vectors that will uncover any defect in chip. Generating effective test patterns efficiently for a digital circuit is thus the goal of

any automatic test pattern generation (ATPG) system. In this paper our discussion based upon the types of ATG methods

3.1 Gate-Level test Generation

The most studied approaches to test generation employ gate level structural models; nearly all commercial test generators do so. The most widely known gate-level test generation algorithms are the D-algorithm and PODEM (Path Oriented Decision Making) [12].If a line in a circuit is 0 (1) when it should be 1 (0), the error signal value on that line is represented by the symbol D (D) for discrepancy. The D-Algorithm uses a greedy value assignment policy—it assigns signal values at the earliest opportunity. This reduces the number of signal evaluations but this makes the decision-making more vulnerable to conflicts and hence increases backtracking. The PODEM test generation algorithm avoids this problem by backtracking only at primary inputs. PODEM does not justify internal values explicitly, as in the D-algorithm. To satisfy an internal objective such as a D or D on some internal line, a value is assigned to a primary input and the circuit is simulated. If the simulation proves that the assignment does not satisfy the objective, PODEM assigns another input value. If during simulation, two values conflict on a line, the algorithm backtracks by changing the value of the last assigned input. When both values have been tried unsuccessfully, the algorithm backtracks to the next-to-last assigned input. In this way, PODEM can exhaustively explore all possible circuit states, but only implicitly [13].A number of test generation techniques have been developed that extend PODEM. Their goal is to reduce the number of backtracks by identifying choices a test generation algorithm might make that cannot lead to a solution, without actually pursuing every decision.

High-level test generation high complexity of gate-level test generation and the hierarchical nature of the design process, several high-level or functional test generation methods have been introduced.. High-level test generation has the following advantages:

- *Fast module evaluation:* Since modules are described at the functional level, they can be evaluated faster than their gate-level equivalents.
- .• *High-level implication:* Implication at the high level may lead to finding values of signals
- *Unique sensitization:* At the high level, efficient procedures can be developed to determine the signals necessary to propagate fault effects at the inputs of a high level module to its outputs. So the propagation of check routine may also be developed to anticipate conflicts earlier and hence reduce the number of backtracks.
- *Reduced backtracking:* This is due to the following: (1) high-level descriptions enclose reconvergent fan-out and hence leads to fewer poor decisions, and (2) module-level decision making leads to improved global implication and consequently conflicts are detected earlier and alternatives are tried sooner.[14][15]

3.2 On-Line Testing

On-line testing addresses the detection of operational faults, and is found in computers that support critical or high-availability applications. The aim of on-line testing is to detect fault effects, that is, errors, quickly and take appropriate corrective action. For example, in some safety-critical applications, the computer system is shut down after an error is detected. In other applications, error detection triggers a reconfiguration mechanism that allows the system to continue its operation, perhaps with some degradation in performance. On-line testing can be performed by external or internal monitoring using either hardware or software; internal monitoring is referred to as self-testing. Monitoring is internal if it takes place on the same substrate as the circuit under test (CUT). This is

Usually considered to be inside an IC.

4. Conclusion

In this research paper we have present a research in fault modelling and test generation for hardware-software covalidation. The growing researchers begin to identify and solve the problems. Covalidation has developed industrial tools point are available which offer the practical solution for test generation. Automation tools are available but designers are not trusted. So a important amount of manual test generation is required for majority of design projects. By examining the state of previous work we can identify areas which should be studied in future work in order to increase the industrial acceptance of covalidation techniques. Hardware-software covalidation is extended from previous research in the hardware and software domains, but communication between hardware and software components is a problem to hardware-software covalidation. A great deal of research in hardware-software covalidation is extended from previous research in the hardware and software domains, but communication between hardware and software components is a problem unique to hardware-software covalidation. The hardware –software introduce the issues of new design so the errors is occurred. Hardware-software communication increased the communication complexity because the interprocessor communication is more difficult in hardware as compare to software. Although the implementation of each primitive may be known to be correct, the primitive itself may be used incorrectly by the designer, resulting in design errors. We can expect to large increases in covalidation productivity through the automation of test Generation.

References

[1] Q. Zhang and I.G. Harris, “A validation fault model for timing-induced functional errors,” in *International Test conference*, October 2001.

[2] S. Arekapudi, F. Xin, J. Peng and I.G. Harris, "Test pattern generation for timing –induced functional errors in hardware-software systems," in *High level Design Validation and Testing Workshop*, 2000.

[3] G. van Brakel,U. Glaiser,and H.G. Kerkhofr and H.T. Vierhaus "Gate Delay Fault Test Generation for Non-Scan Circuits" IEEE 1995 pp. 308-312.

[4] Ian G. Harris "Fault Models and Test Generation for Hardware-Software Covalidation" National Science Foundation under grant number 0204134

[5] T. J. McCabe, "A complexity measure", *IEEE Transactions on Software Engineering*, vol. SE-2, pp. 308–320, December 1976.

[6] R. Bailey, *Human Error in Computer Systems*, Prentice-Hall, Englewood Cliffs, N. J., 1981.

[7] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley, Reading, Mass., 1989.

[8] R. H. Untch, A. J. Offutt, and M. J. Harrold, "Mutation analysis using mutant schemata", *Proc. International Symposium on Software Testing, Analysis, and Verification*, 1993, pp. 139-148.

[9] S. Dey, A. Raghunathan, and K. D. Wagner, "Design for testability techniques at the behavioral and register-transfer level", *Journal of Electronic Testing: Theory and Applications(JETTA)*, vol. 13, pp. 7–19, October 1998.

[10] B. Beizer, *Software Testing Techniques, Second Edition*, VanNostrand Reinhold, 1990.

[11] P. Narain et al., "A high-level approach to test generation", *IEEE Transactions on Circuits and Systems. Part I, Fundamental Theory and Applications*, Vol. 40, pp. 483-492, July 1993.

[12] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic design verification via test generation", *IEEE Transactions on Computer-Aided Design*, Vol. 7, pp. 138-148,January 1988.

[13] J. D. Calhoun and F. Brglez, "A framework and method for hierarchical test generation", *IEEE Transactions on Computer-Aided Design*, Vol. 11, pp. 45-67,January 1992.

[14] M. S. Abadir and H. K. Reghbati, "Functional testing of semiconductor random access memories", *Computing Surveys*, Vol. 15, No. 3, September 1983.

[15] D. D. Gajski and F. Vahid, "Specification and design of embedded hardware-software systems", *IEEE Design and Test of Computers*, vol. 12, pp. 53–67, 1995